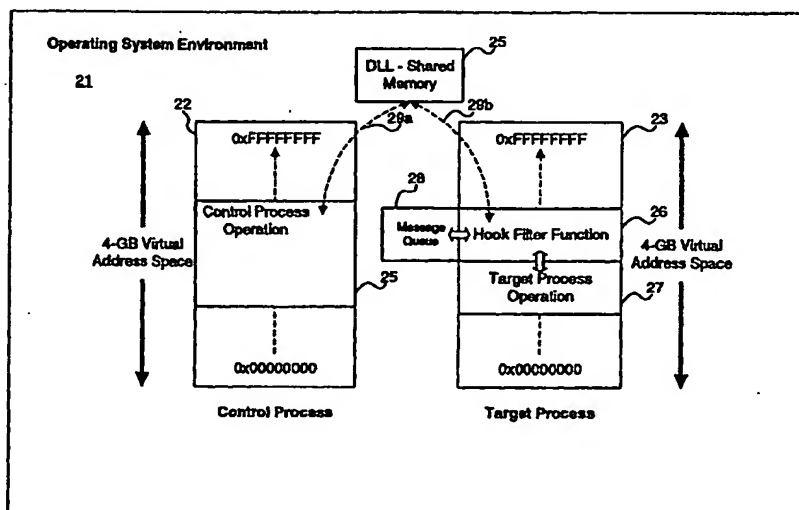




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁷ : G06F 9/46		A1	(11) International Publication Number: WO 00/13085
			(43) International Publication Date: 9 March 2000 (09.03.00)
(21) International Application Number: PCT/US99/19968		(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).	
(22) International Filing Date: 31 August 1999 (31.08.99)			
(30) Priority Data: 60/098,762 1 September 1998 (01.09.98) US 60/100,081 14 September 1998 (14.09.98) US		Published <i>With international search report. Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>	
(71) Applicant: LERNOUT & HAUSPIE SPEECH PRODUCTS N.V. [BE/BE]; Flanders Language Valley 50, B-8900 Leper (BE).			
(71)(72) Applicant and Inventor: BUNCE, Erik [US/US]; 176 Acton Road, Stow, MA 01775 (US).			
(72) Inventor: DINGER, Tom; 38 Cedarcroft Lane, Waltham, MA 02451-2116 (US).			
(74) Agents: SUNSTEIN, Bruce, D. et al.; Bromberg & Sunstein LLP, 125 Summer Street, Boston, MA 02110-1618 (US).			

(54) Title: INTERFACE BETWEEN A CONTROL PROCESS AND A TARGET PROCESS



(57) Abstract

Indirect access and control of a target computer process from a control computer process in a multitasking computer operating system environment is described. The control process contains a dynamic linked library (DLL) file. A hook filter function is then inserted into the target process using one of the operating system commands. This forcibly maps the control process DLL into the target process memory space. The hook filter function monitors all messages from the operating system which enter the target process via its message queue. The control process places necessary arguments and data in the DLL and sends a message to the hook filter function requesting an operation within the target process. The hook filter function recognizes the message as being from the control process and that the arguments and data are in the DLL. Data may also be passed from the target process via the DLL to the control process.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

Interface Between a Control Process and a Target Process

Technical Field

The present invention relates to a method of indirectly accessing and
5 controlling one process from another within a multitasking computer operating system environment.

Background Art

In older 16-bit personal computer operating system environments such as
10 Windows 3.1® available from Microsoft Corporation, Redmond, Washington, all running programs—known as processes—share a single virtual address space. Any process can read from and write to memory belonging to any other process, even memory used by the operating system. If a first process inadvertently overwrites data being used by another running second process, the second
15 process could well become unstable and crash. With the advent of 32-bit personal computer operating systems, more virtual address space is available and a more robust environment may be created which avoids this problem.

In some 32-bit computer system operating environments such as Windows 95®, Windows 98®, and Windows NT®, all available from Microsoft
20 Corporation, Redmond, Washington, each process is assigned a unique process address space, typically, 4-GB. Such a process address space contains the executable code and data necessary for the process, that is, the code for the EXE and DLL files and their required data. In addition to an address space, a process also owns certain resources such as files, dynamic memory allocations, and
25 threads. Within a given process, when pointers are used to reference memory, the pointer value refers to a memory address in that process's own assigned memory space. It is not possible for one process to create a pointer which references memory assigned to another process. As a result, the system is stable and robust, and less likely to crash due to memory conflicts. However, it is also
30 very difficult for one process to interact with and affect another.

For example, when a user is prompted for an input by a dialog box, the text from the user is typically stored in an edit control. An edit control is simply a window used to store free-form text input by a user. Within a process, accessing a given edit control is almost trivial. Simply obtaining the handle (or
5 name) of a given edit control window allows access via the operating system's control window messages. However, the most common such control window messages do not work across process boundaries in a 32-bit system with separate virtual address spaces for each process. This is because use of control windows messages typically requires that some message argument parameters be in the
10 form of a memory address of a required data structure. As described above, however, memory addresses are only meaningful within a given process. Thus, user text within an edit control of one process is not available to another process.

To enable a speech recognition system to control a target process when both are running in an advanced 32-bit windowing computer, it is easy if the
15 target application is written with features of the controlling speech recognition system in mind. However, for a speech recognition system to control a target process that is not thus speech-enabled requires utilizing whichever suite of characteristics the target process exhibits in relation to the operating system. These characteristics often make sophisticated control by a speech recognition
20 system, at best, very difficult.

Summary of the Invention

A preferred embodiment of the present invention provides a method for interfacing between a control process and a target process in a multitasking
25 computer operating system environment. The method includes the steps of :
a. establishing a region of shared memory between the control process and the target process;
b. generating a request with the control process for a desired operation by the target process and storing arguments and data for the request
30 in the region of shared memory;

c. communicating the request for the desired operation to the target process; and

d. effecting execution of the desired operation by the target process.

In another embodiment, the method includes:

5 a. establishing a region of shared memory between the control process and the target process;

b. hooking an executable link module into an address space of the target process;

c. generating a request for a desired operation by the target process
10 and storing arguments and data for the request in the shared memory;

d. posting a private message from the control process to the target process notifying the executable link module of the request;

e. receiving the private message in the executable link module;

f. executing the requested action in the target process; and

15 g. sending a private message acknowledgment from the executable link module to the control process.

In such an embodiment, the step of receiving may further include retrieving arguments and data for the request from the shared memory. In addition, or alternatively, the step of sending may further include writing new
20 data from the target process into the shared memory. In such an embodiment, the step of sending may further include retrieving the new data by the control process from the shared memory.

A preferred embodiment is also directed to an interface system used between a control process and a target process in a multitasking computer
25 operating system environment. Such a system comprises:

a. a region of shared memory that is established between the control process and the target process;

b. a request generator which generates a request from the control process for a desired operation by the target process and stores arguments and
30 data for the request in the region of shared memory;

c. a request communicator which communicates the request for the desired operation to the target process; and

d. a request receiver which effects execution by the target process of the desired operation.

5 Another preferred embodiment includes an interface system used between a control process and a target process in a multitasking computer operating system environment. Such a system comprises:

a. a region of shared memory that is established between the control process and the target process;

10 b. an executable link module hooked into an address space of the target process;

c. a request generator which generates a request for a desired operation by the target process and stores arguments and data for the request in the shared memory;

15 d. a request communicator which communicates the request for the desired operation to the executable link module;

e. receiving the private message in the executable link module;

f. means for executing the requested action in the target process; and

20 g. a message acknowledger within the executable link module which sends a message acknowledging receipt and accomplishment of the request by the target process to the control process.

Brief Description of the Drawings

The present invention will be more readily understood by reference to the
25 following detailed description taken with the accompanying drawings, in which:

Fig. 1 illustrates the logical flow associated with a preferred embodiment.

Fig. 2 illustrates the inter-relationships of various functional blocks within the operating system environment according to a preferred embodiment.

Fig. 3 illustrates computer process arrangements in which a preferred
30 embodiment may be employed.

Detailed Description of Specific Embodiments

A preferred embodiment of the present invention, as shown in Figs. 1-3, includes a technique for an originating control process 24 having generally a non-keyboard input 32, to exchange detailed information with and send
5 commands from a command output block 31 to a separate target process 27. Such techniques are useful, for instance, where the control process 24 is an automatic speech recognition engine 33 having a speech input 34, and running in a 32-bit operating system environment 21, such as Windows NT®, and the target process 27 is a non-speech aware application, such as an e-mail program,
10 running separately in the same operating system environment 21. It may be desirable for the speech recognition control process 24 to obtain access to and control various forms of data in another target process 27 ranging from simple text to more complex data structures such as in a database file. A preferred embodiment may be advantageously employed in various operating system
15 environments 21, including, for instance, Windows NT®, Windows 95®, Windows 98®, X for Windows, and UNIX.

A preferred embodiment involves inserting a hook filter function 26 (also known as a callback function) into the target process 27, and creating a pointer 29b from the hook filter function 26 to an executable link module in shared
20 system memory, such as a dynamic linked library (DLL) file 25. A hook filter function 26 is a program mechanism which can intercept events in a process message queue 28 such as messages, keystrokes, and mouse actions before they reach the target process 27. To operate between separate processes, the filter function invoked by the hook filter function 26 is placed within a DLL 25. The
25 DLL 25 then allows inter-process communication and control via shared memory within the DLL 25. Thus, the shared memory 25 is common to the operating system 21 and accessible to both the control process 24 via pointer 29a, and the target process 27 via pointer 29b. This technique allows for very efficient transfer of information between the control process 24 and the target

process 27. In addition, the control process 24 may obtain access to information from the target process 27 which is not usually available via inter process communication (IPC), such as user input text in the edit control of a dialog box of a target process 27. It also allows text from the control process 24, e.g., the text
5 output from an automatic speech recognition program, to be passed, via the shared memory in the DLL 25, to an edit control in the target process 27 which treats the text as if it were provided by the user from a keyboard.

Such inter-process text passing to a target process edit control may be considered a short term or an immediately obtainable advantage of a preferred
10 embodiment of the present invention. More prolonged or longer term usefulness may also be realized. For example, the hook filter function 26 may be employed for ongoing monitoring of the operation of the target process 27 for the occurrence of target process events which are of interest to the control process 24. Such interesting events may include, for example, opening a menu,
15 making a menu selection, or opening a dialog box within the target process 27.

A preferred embodiment may start with the control process 24 installing a WH_GETMESSAGE hook filter function 26 in the target process 27, step 11 in Fig. 1. The hook filter function 26 is installed by calling *SetWindowsHookEx* as follows:

20

```
HHOOK hhook = SetWindowsHookEx(WH_GETMESSAGE, GetMsgProc, hinstDll,  
TargetThreadName);
```

The first parameter, WH_GETMESSAGE, indicates the type of hook filter
25 function 26 to install. WH_GETMESSAGE is a hook filter function 26 called when the *GetMessage* or the *PeekMessage* function is about to return a message to the target process 27. The hook filter function 26 will receive a pointer 29b to a message structure within the shared memory of the DLL 25 which contains the actual message. The second parameter, *GetMsgProc*, identifies the address of a
30 function in the shared memory of the DLL 25 that the operating system 21

should call whenever a window in the target process 27 is about to process a message in its message queue 28. The third parameter, *hinstDll*, identifies the DLL 25 that contains the *GetMsgProc* function. In Windows NT, a DLL's *hinstDll* value identifies the 32-bit virtual memory address where the DLL 25 is mapped
5 into the address space 22 of the control process 24. Since the memory address space of the DLL 25 will be shared by both the control process 24 and the target process 27, it may be considered accessible to each process via a pointer, 29a and 29b respectively. The fourth parameter, *TargetThreadName*, identifies the thread to hook in the target process 27. A value of NULL may be passed for the
10 fourth parameter to instruct the operating system 21 to hook all threads in the system.

When *GetMessage* or *PeekMessage* is about to return a message to the target process 27 from the message queue 28, the operating system 21 checks whether a WH_GETMESSAGE hook 26 is installed on that thread in the target process 27,
15 and whether the DLL 25 containing the *GetMsgProc* function is mapped into the address space 23 of the target process 27. If the DLL 25 has not been mapped, the operating system 21 forces the DLL 25 to be mapped into the address space 23 of the target process 27 via pointer 29b and increments a lock count on the mapping of the DLL 25 in the target process 27. The operating system 21 looks
20 at the *hinstDll* of the DLL 25 as it applies to the target process 27 and checks to see whether the *hinstDll* of the DLL 25 is at the same location as it applies to the control process 24. If the *hinstDlls* are the same, the memory address of the *GetMsgProc* function is also the same in the two process address spaces. In such a case, the operating system 21 can simply call the *GetMsgProc* function in the
25 address space 22 of the control process 24.

If the *hinstDlls* are different, the operating system 21 must determine the virtual memory address of the *GetMsgProc* function in the address space 23 of the target process 27 according to the following formula:

$$\text{GetMsgProcTarget} = \text{hinstDllTarget} + (\text{GetMsgProcControl} - \text{hinstDllControl})$$

Subtracting *hinstDllControl* from *GetMsgProcControl* results in the offset in bytes for the *GetMsgProc* function. Adding this offset to *hinstDllTarget* gives the location of the *GetMsgProc* function as it applies to the mapping of the DLL 25 in the address space 23 of the target process 27. The operating system 21

- 5 increments a lock count on the mapping of the DLL 25 in the address space 23 of the target process 27, and also calls the *GetMsgProc* function in the address space 23 of the target process 27. When *GetMsgProc* returns, the operating system 21 decrements a lock count on the mapping of the DLL 25 in the target process 27.

- Injecting or mapping the DLL 25 containing the hook filter function 26
- 10 maps the whole DLL 25 into the target process 27, not just the hook filter function 26. Any and all functions contained in the DLL 25 now exist and can be called from threads running in the context of the target process 27. For instance, to subclass a window created by a thread in the target process 27, the WH_GETMESSAGE hook 26 is set on the thread that created the window, and
- 15 then, when the *GetMsgProc* function is called, call *SetWindowLong* to subclass the window. Of course, the subclass procedure must be in the same DLL 25 as the *GetMsgProc* function.

- The hook filter function 26 examines messages in the message queue 28 coming into the target process 27 for messages from the control process 24.
- 20 Thereafter, the control process 24, in step 12, may generate a request for action or information from the target process 27 and via the pointer 29a place in the shared memory of the DLL 25 a description of the type of operation requested and any arguments necessary. Then, in step 13, a message is sent from the control process 24 to the target process 27 notifying the hook filter function 26 of
- 25 the request from the control function 24. This message includes an index pointer, 29b in Fig. 2, to the shared memory location in the DLL 25 where the required information and arguments are available. The hook filter function 26 in the target process 27, in step 14, receives the message in the address space 23 of the target process 27, recognizes the message as a command from the control

process 24, and uses the index pointer 29 to obtain the necessary information and arguments from the shared memory of the DLL 25, step 15.

The target process 27, in step 16, then makes calls to the operating system 21 accessing the pertinent window to perform the required operation. As a
5 result of this sequence, the control process 24 enjoys the same level of access within the target process 27 as does the target process 27 itself. The control process 24 may thereafter perform any actions within the target process 27 which the target process 27 itself could do. Thus, the message may direct the target
10 process 27 to, for example, move the insertion point, or retrieve text or data from an edit control in the target process 27 and, optionally as in step 17, write the data to the shared memory of the DLL 25.

After the control process 24 has sent its initial message to the hook filter function 26, it waits for notification from the target process 27 that it has accomplished the directed operation. This notification, according to step 18 of
15 Fig. 1, may be accomplished in various manners such as by posting a message in the message queue of the control process 24, or by changing the state of an event object which managed by the operating system 21 and monitored by the control process 24. Any text or data from the target process 27 may be posted to the shared memory of the DLL where it may be retrieved by the control process 24,
20 step 19 of Fig. 1. If the control process 24 no longer requires the use of the hook filter function 26, the hook filter function 26 may be removed from the target process 27 by the control process 24. This may be accomplished by use of the *UnhookWindowsHookEx* function.

What is claimed is:

1. A method for interfacing between a control process and a target process in a multitasking computer operating system environment, the method
5 comprising:
 - a. establishing a region of shared memory between the control process and the target process;
 - b. generating a request with the control process for a desired operation by the target process and storing arguments and data for the request
10 in the region of shared memory;
 - c. communicating the request for the desired operation to the target process; and
 - d. effecting execution of the desired operation by the target process.
- 15 2. A method for interfacing between a control process and a target process in a multitasking computer operating system environment, the method comprising:
 - a. establishing a region of shared memory between the control process and the target process;
 - 20 b. hooking an executable link module into an address space of the target process;
 - c. generating a request for a desired operation by the target process and storing arguments and data for the request in the shared memory;
 - d. posting a private message from the control process to the target
25 process notifying the executable link module of the request;
 - e. receiving the private message in the executable link module;
 - f. executing the requested action in the target process; and
 - g. sending a private message acknowledgment from the executable link module to the control process.

30

3. A method as in claim 2, wherein the step of receiving further includes retrieving arguments and data for the request from the shared memory.
4. A method as in claim 2, wherein the step of sending further includes
5 writing new data from the target process into the shared memory.
5. A method as in claim 4, wherein the step of sending further includes retrieving the new data by the control process from the shared memory.
- 10 6. An interface system used between a control process and a target process in a multitasking computer operating system environment, the system comprising:
 - a. a region of shared memory that is established between the control process and the target process;
 - 15 b. a request generator which generates a request from the control process for a desired operation by the target process and stores arguments and data for the request in the region of shared memory;
 - c. a request communicator which communicates the request for the desired operation to the target process; and
 - 20 d. a request receiver which effects execution by the target process of the desired operation.
7. An interface system used between a control process and a target process in a multitasking computer operating system environment, the system
25 comprising:
 - a. a region of shared memory that is established between the control process and the target process;
 - b. an executable link module hooked into an address space of the target process;

- c. a request generator which generates a request for a desired operation by the target process and stores arguments and data for the request in the shared memory;
- d. a request communicator which communicates the request for the
5 desired operation to the executable link module;
- e. receiving the private message in the executable link module;
- f. means for executing the requested action in the target process; and
- g. a message acknowledger within the executable link module which
sends a message acknowledging receipt and accomplishment of the request by
10 the target process to the control process.

[96358]

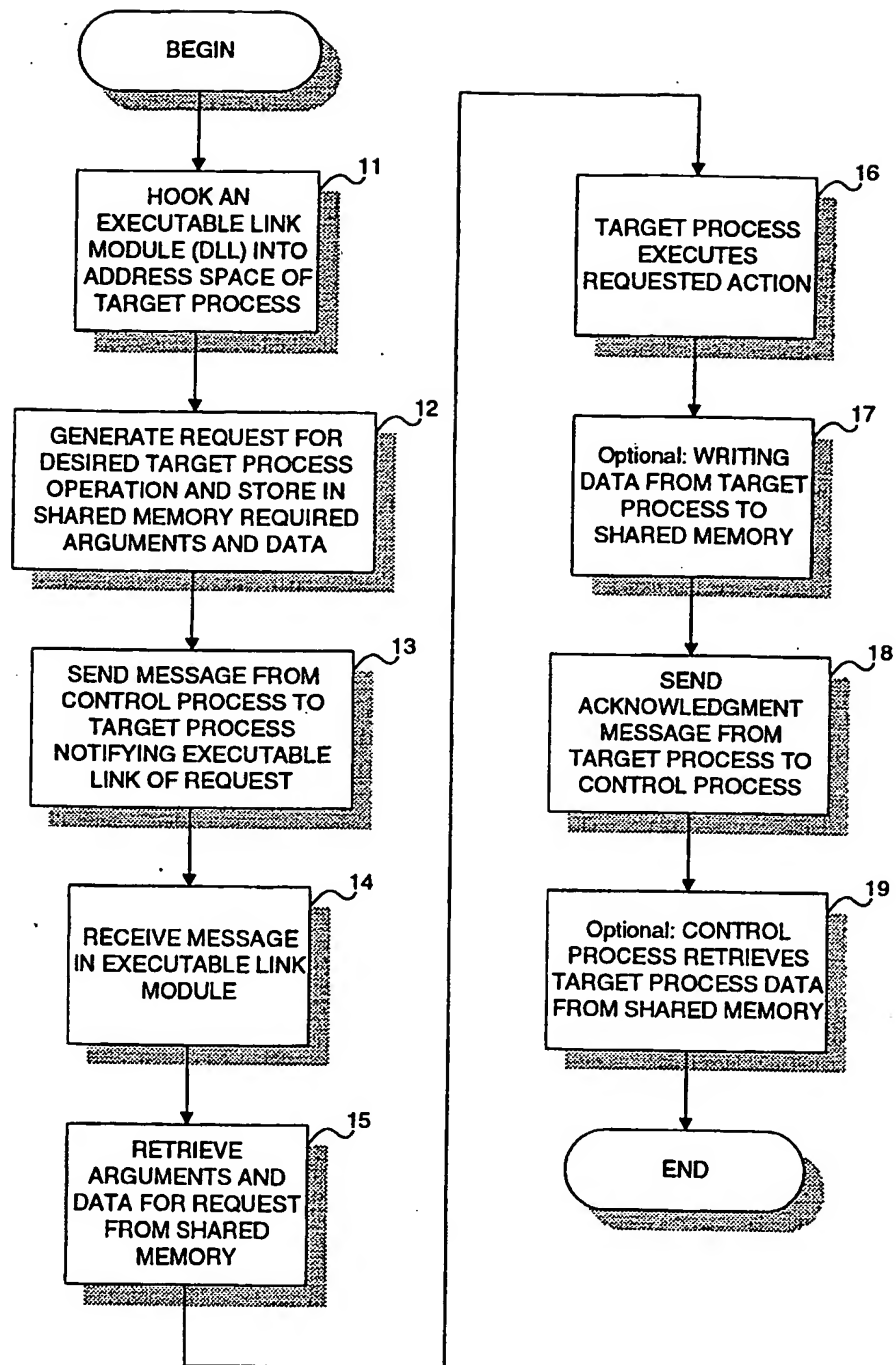


FIGURE 1

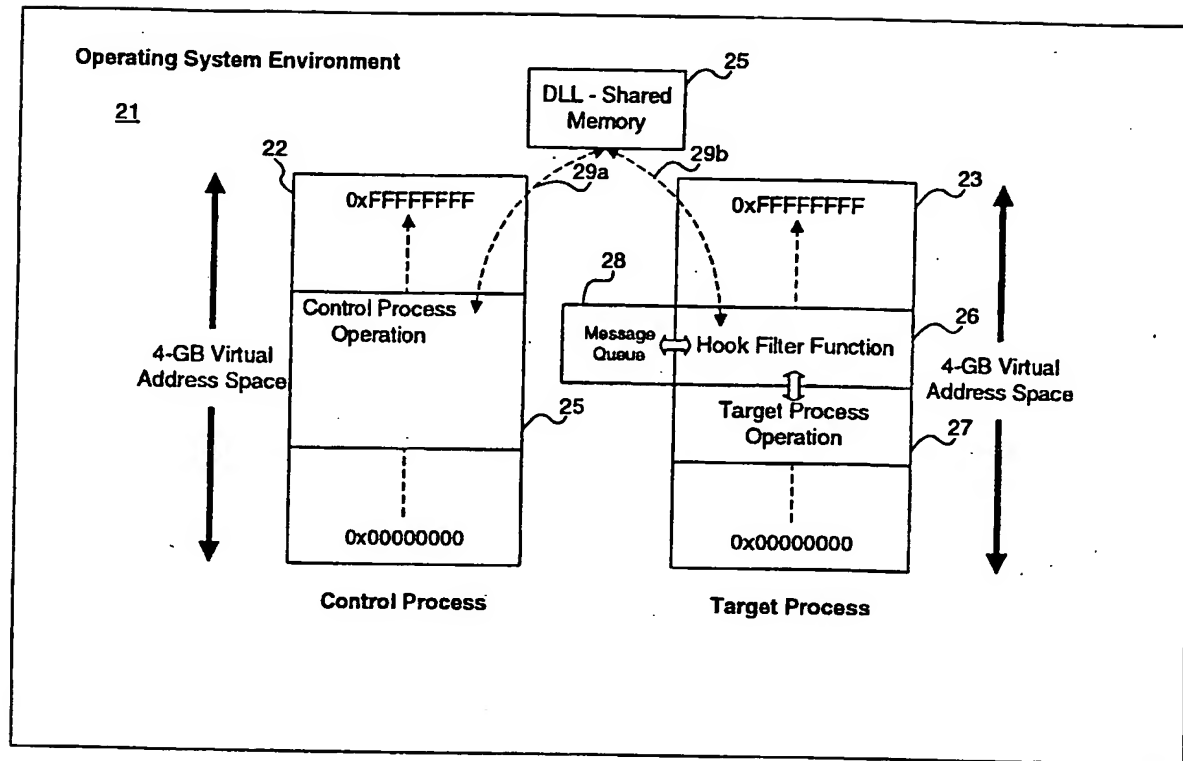


Figure 2

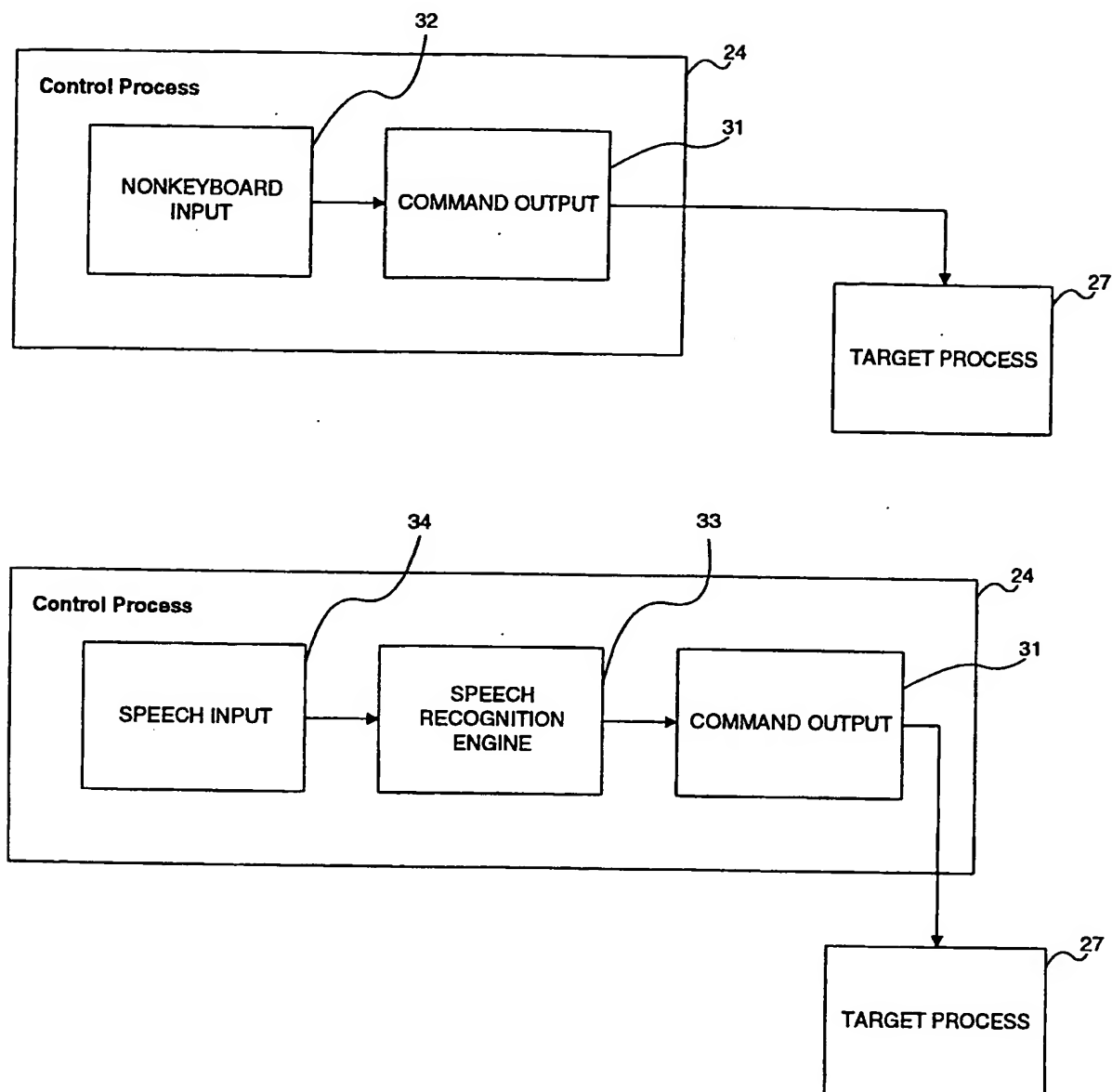


Figure 3

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 99/19968

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 647 902 A (IBM) 12 April 1995 (1995-04-12)	1,6
A	page 3, line 21 -page 5, line 5; figures 1-3	2-5,7
A	--- "METHOD FOR IMPLEMENTING DIRECTORY-LIMIT FUNCTIONS IN WINDOWS FILE MANAGER" IBM TECHNICAL DISCLOSURE BULLETIN, US, IBM CORP. NEW YORK, vol. 38, no. 7, July 1995 (1995-07), page 9-10 XP000521585 ISSN: 0018-8689 the whole document --- -/-	2,7

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search

28 January 2000

Date of mailing of the international search report

03/02/2000

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Michel, T

INTERNATIONAL SEARCH REPORT

Inter national Application No

PCT/US 99/19968

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	<p>JORDAN D: "INSTANTIATION OF C++ OBJECTS IN SHARED MEMORY" JOURNAL OF OBJECT-ORIENTED PROGRAMMING, US, NEW YORK, NY, vol. 4, no. 1, March 1991 (1991-03), page 21-24, 26-28 XP000501371 the whole document</p>	1-7

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 99/19968

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0647902 A	12-04-1995	DE 69418963 D	15-07-1999
		JP 7121373 A	12-05-1995
		US 5737605 A	07-04-1998